IAccess Datenbanksystem

Semesterarbeit Objektorientiertes Programmieren 2

Inhaltsverzeichnis

[Abbildungsverzeichnis 3](#_Toc82455726)

[Tabellenverzeichnis 3](#_Toc82455727)

[1 Einleitung 4](#_Toc82455728)

[2 Aufgabenstellung 4](#_Toc82455729)

[2.1 Umschreibung der Aufgabenstellung 4](#_Toc82455730)

[3 Projektumfeldanalyse¨ 4](#_Toc82455731)

[3.1 Stakeholder 5](#_Toc82455732)

[3.2 Situationsanalyse Seeland-Versicherung 5](#_Toc82455733)

[4 Ziele 5](#_Toc82455734)

[4.1 MUSS-Ziele 5](#_Toc82455735)

[4.2 KANN-Ziele 6](#_Toc82455736)

[5 Anforderungen 6](#_Toc82455737)

[5.1 Funktionale Anforderungen 6](#_Toc82455738)

[5.2 Nicht-funktionale Anforderungen 6](#_Toc82455739)

[6 Situationsanalyse Seeland Versicherung 7](#_Toc82455740)

[6.1 Kontextdiagram 7](#_Toc82455741)

[6.2 Use-Case Diagramme 8](#_Toc82455742)

[7 Konzept 9](#_Toc82455743)

[7.1 Klassendiagram 9](#_Toc82455744)

[7.2 Sequenzdiagram 10](#_Toc82455745)

[7.2.1 Erstellen eines neuen Record 10](#_Toc82455746)

[7.2.2 Select eines Records 10](#_Toc82455747)

[7.2.3 Update eines Records 11](#_Toc82455748)

[7.2.4 Löschen eines Records 12](#_Toc82455749)

[8 Realisierung 13](#_Toc82455750)

[8.1 Versionierung 13](#_Toc82455751)

[8.2 Speichern der Daten 13](#_Toc82455752)

# Abbildungsverzeichnis

[Abbildung 1: Identifizierte Stakeholder 5](#_Toc81852871)

[Abbildung 2: Kontextdiagram Seefeld-Versicherung 7](#_Toc81852872)

[Abbildung 3: Schadensmeldung Kunde 8](#_Toc81852873)

[Abbildung 4: Schadensmeldung Mitarbeiter 8](#_Toc81852874)

[Abbildung 5: Use-Case Kundendaten 9](#_Toc81852875)

[Abbildung 6: UML-Klassendiagram zu iAccess 9](#_Toc81852876)

[Abbildung 7: UML-Sequenzdiagram zum Erstellen eines neuen Records. 10](#_Toc81852877)

[Abbildung 8: UML-Sequenzdiagram zum Selektieren eines Records. 10](#_Toc81852878)

[Abbildung 9: UML-Sequenzdiagram zum Updaten eines Records. 11](#_Toc81852879)

[Abbildung 10: UML-Sequenzdiagram zum Löschen eines Records. 12](#_Toc81852880)

# Tabellenverzeichnis

[Table 1: MUSS-Ziele 6](#_Toc81852898)

[Table 2: KANN-Ziele 6](#_Toc81852899)

[Table 3: Funktionale Anforderungen 6](#_Toc81852900)

[Table 4: Nicht-funktionale Anforderungen 7](#_Toc81852901)

# Einleitung

Severin Gafner und David Hänni führen im Rahmen des Studiums zum Techniker HF Applikationsentwicklung für das Modul Objektorientiertes Programmieren 2 eine Semesterarbeit durch.

# Aufgabenstellung

Nachfolgend die Umschreibung der Aufgabenstellen 1:1 aus dem Auftrag kopiert.

## Umschreibung der Aufgabenstellung

Wir sind die IT-Firma NewIdeas und wir haben die nachfolgende Geschäftsidee:

Im Zusammenhang mit Automationsvorhaben werden von unseren Kunden immer mehr kleine aber individuelle Lösungen gesucht.

Wir haben nun festgestellt, dass sich die bekannten (umfangreichen) Datenbanksysteme für derartige Vorhaben nicht oder nur bedingt eignen, vor allem die teilweise abstrus teuren Lizenzmodelle lassen diese als nur bedingt tauglich erscheinen.

Die verfügbaren OpenSource/Freewarelösungen können wir nicht ernst nehmen resp. wollen wir aus Gründen der Nachhaltigkeit nicht verwenden.

Wir haben nun vor, eine eigene relationale Datenbank aufzubauen, diese soll vor allem für den geschilderten Kontext verwendet werden und dabei die nachfolgenden Rahmenbedingungen erfüllen:

# Projektumfeldanalyse¨

Mit der Projekumfeldanalyse werden möglichst viele Information über vorhandene Interessen, Bedürfnisse, Einussmöglichkeiten und Beziehungen im Projektumfeld ermittelt.

## Stakeholder

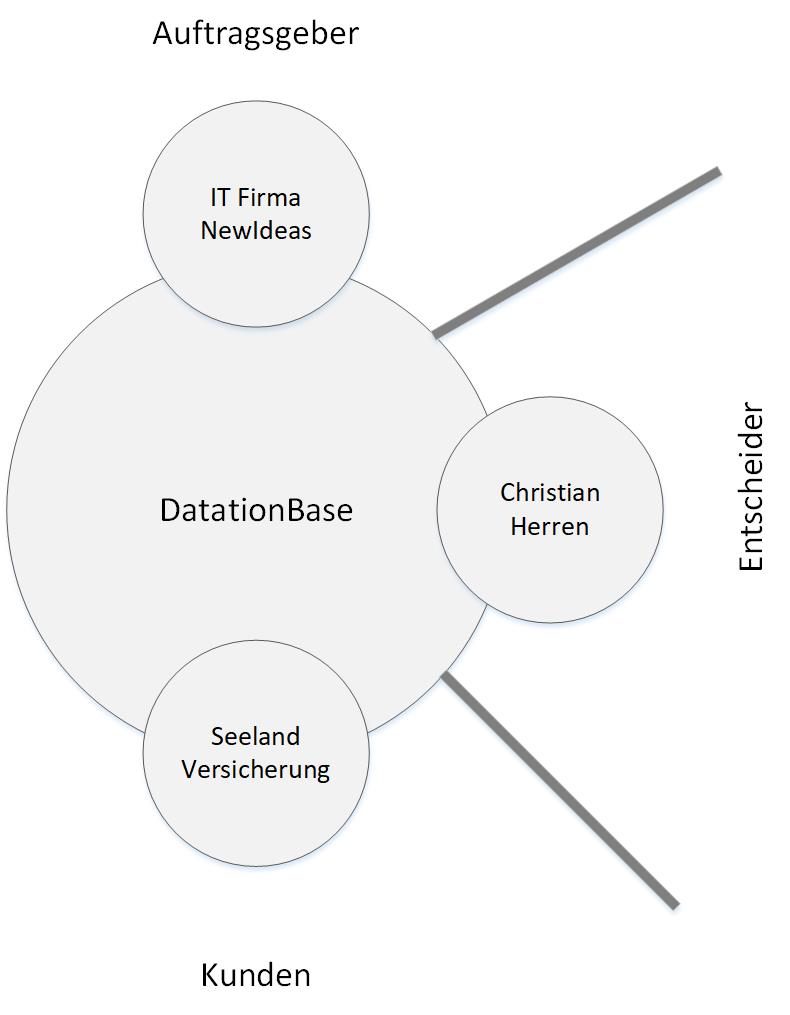


Abbildung 1: Identifizierte Stakeholder

Die Stakeholder wurden in drei Kategorien eingeteilt:

**Auftraggeber** Der Auftraggeber ist die IT Firma NewIdeas. Ihr Ziel ist es, ihre Kunden mit einer eigenen Datenbanklösung an sich zu binden und so zukünftige Wartungen und Modifikationen an der Lösung durchführen zu können.

**Kunden** Als erstes Kunde wird die Seeland-Versicherung die Datenbank einsetzen. Später werden weitere Kunden dazustossen. Der Kunde möchte eine performante Datenbank ohne Datenverlust.

**Entscheider** Christoph Herren ist ebenfalls ein Stakeholder dieses Projektes. Er überwacht die Durchführung der Arbeit und bewertet das Endresultat.

## Situationsanalyse Seeland-Versicherung

Die Seeland-Versicherung spielt als erster Kunde eine sehr wichtige Rolle. Mit den 1'500 Kunden und 450 Schadensfälle pro Jahr beinhaltet die Datenbank nur wenig Datensätze. Angenommen es werden die Schadensfälle der letzten 20 Jahre migriert, so umfasst die Datenbank immer noch nur knapp 10'000 Datensätze.

# Ziele

Die Projektziele zeigen auf, was erreicht werden muss.

## MUSS-Ziele

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Beschreibung** |
| 1 | Anforderungen geklärt | Die funktionalen und nicht-funktionalen Anforderungen an das Produkt sind klar definiert. |
| 2 | Use-Cases definiert | Use Cases wurden nach UML Richtlinien definiert und beschrieben. |
| 4 | Systemarchitektur beschreiben | Die Systemarchitektur und die einzelnen Komponenten sowie deren Schnittstellen wurden beschrieben. |
| 5 | Fertigstellung erreicht | Das Produkt ist einsatzfähig und hat alle Testszenarien vom Testkonzept bestanden. Alle Use Cases wurden implementiert. |

Table 1: MUSS-Ziele

## KANN-Ziele

|  |  |  |
| --- | --- | --- |
| ID | Name | Beschreibung |
| 6 | Datenimport | Über eine Schnittstelle können aus einer Datei direkt in die Datenbank importiert werden. |

Table 2: KANN-Ziele

# Anforderungen

Die Anforderungen an das Produkt werden von den Zielen abgeleitet. Sie werden in funktionale und nicht-funktionale Anforderungen aufgeteilt.

## Funktionale Anforderungen

|  |  |  |
| --- | --- | --- |
| ID | Name | Beschreibung |
| 1 | Tabellen definieren | Es können Tabellen mit unterschiedlichen Namen definiert werden. |
| 2 | Felder definieren | Auf einer Tabelle können Felder erfasst werden. Diese Felder besitzen einen Namen und einen der folgenden Datentypen:   * String * Int * Boolean |
| 3 | Datensätze hinzufügen | Auf einer bestehenden Tabelle können über eine C# Methode mehrere neue Datensätze hinzugefügt werden. |
| 4 | Datensätze löschen | Auf einer bestehenden Tabelle können über eine C# Methode mehrere bestehende Datensätze gelöscht werden. |
| 5 | Datensätze verändern | Auf einer bestehenden Tabelle können über eine C# Methode mehrere bestehende Datensätze verändert werden. |
| 6 | Datensätze anzeigen | Auf einer bestehenden Tabelle können über eine C# Methode mehrere bestehende Datensätze ausgelesen werden. |
| 7 | Schnittstelle XML | Über eine Schnittstelle können Daten im XML Format in die Datenbank importiert werden. |
| 8 | Schnittstelle JSON | Über eine Schnittstelle können Daten im JSON Format in die Datenbank importiert werden. |
| 9 | Schnittstelle CSV | Über eine Schnittstelle können Daten im CSV Format in die Datenbank importiert werden. |

Table 3: Funktionale Anforderungen

## Nicht-funktionale Anforderungen

|  |  |  |
| --- | --- | --- |
| ID | Name | Beschreibung |
| 10 | Speichervolumen | Das Produkt muss mindestens eine 2GB grosse Datenbank mit 1'000'000 Datensätzen bewältigen können. |
| 11 | Zugriffszeiten | Bei 100'000 Datensätzen darf die Zugriffszeit auf einen Datensatz nicht länger als 100 ms dauern. |
| 12 | Programmiersprache | Die Datenbank wird mittels C# implementiert. |
| 13 | Projektabgabe | Das Produkt sowie die Projektdokumentation müssen bis am 28.09.2021 um 18:30 Uhr abgegeben sein. |

Table 4: Nicht-funktionale Anforderungen

# Situationsanalyse Seeland Versicherung

## Kontextdiagram

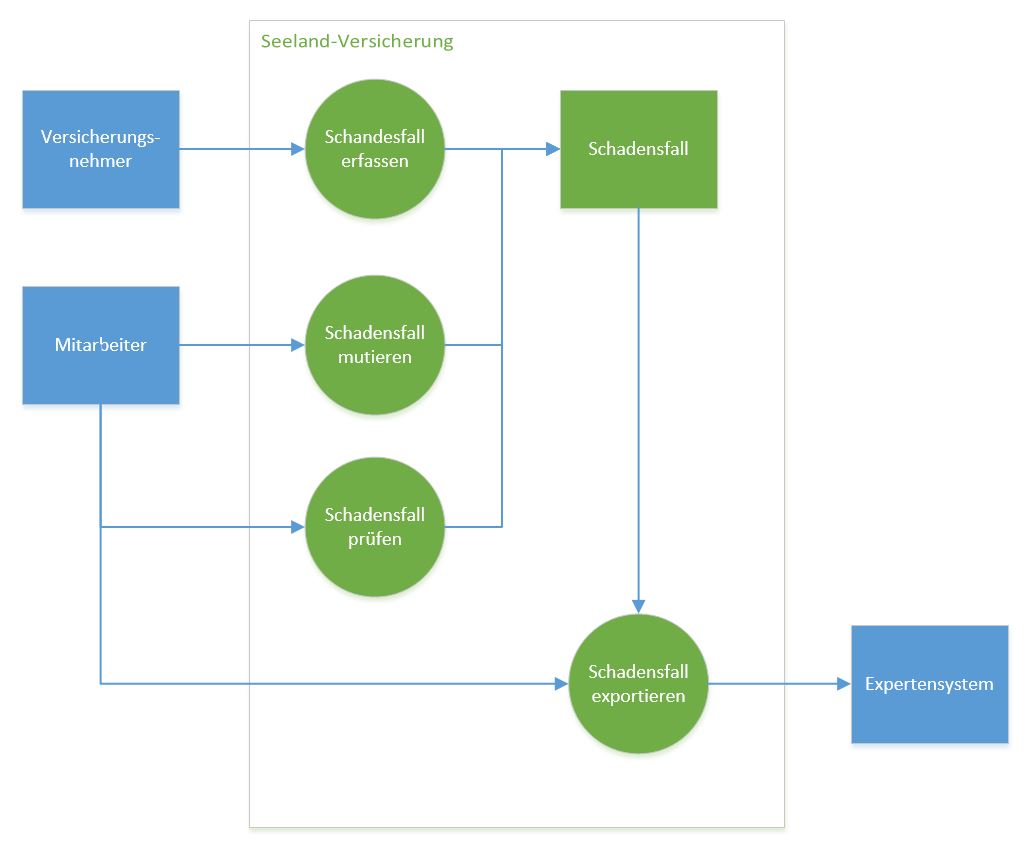


Abbildung 2: Kontextdiagram Seefeld-Versicherung

## Use-Case Diagramme

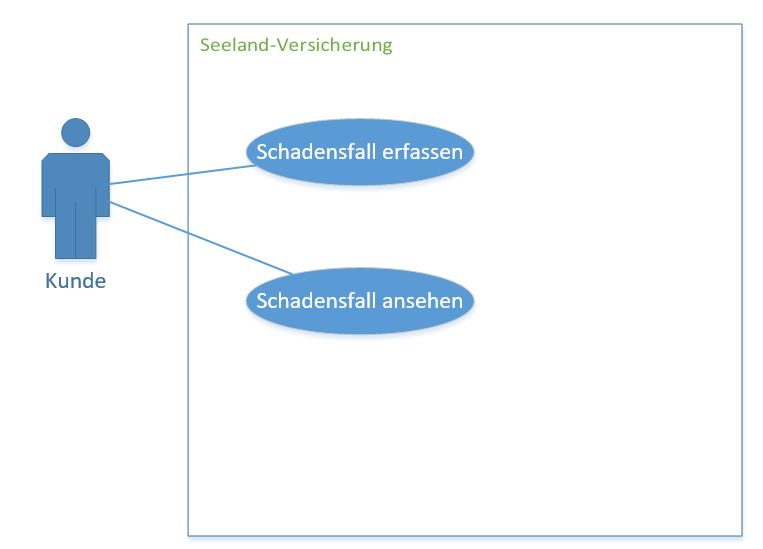


Abbildung 3: Schadensmeldung Kunde

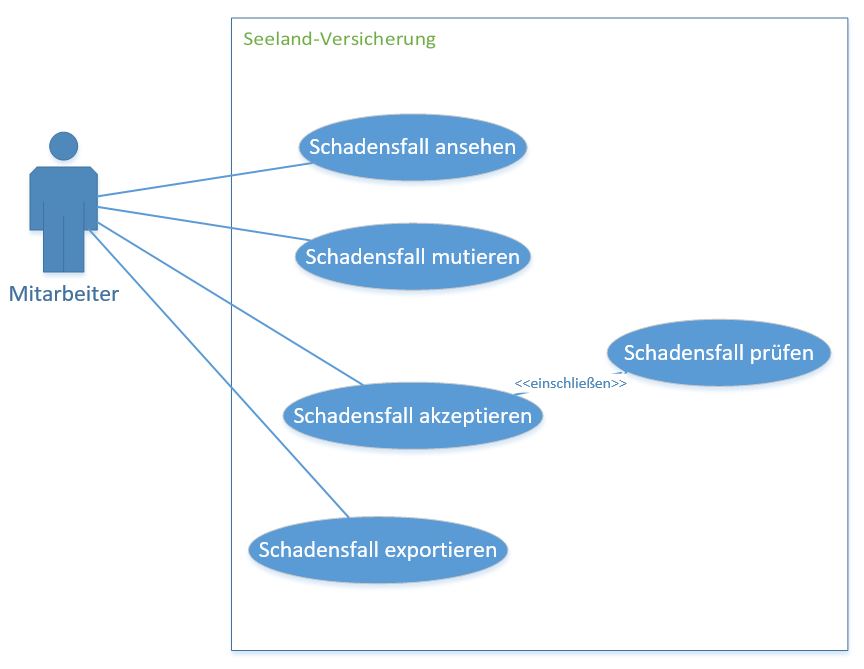


Abbildung 4: Schadensmeldung Mitarbeiter

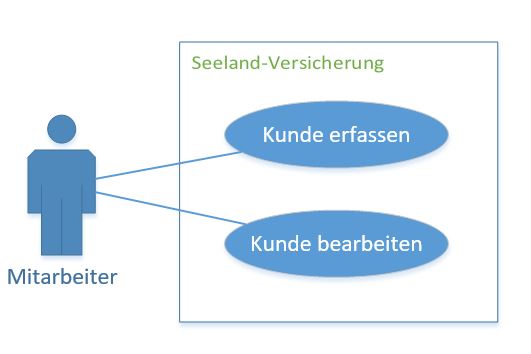


Abbildung 5: Use-Case Kundendaten

# Konzept

## Klassendiagram

Das Klassendiagram zeigt die Klassen von IAccess. Das Third Party Package muss eine eigene Klasse erstellen, welche von Record erbt. In dieser Klasse wird die Datenstruktur dieses Records festgehalten.

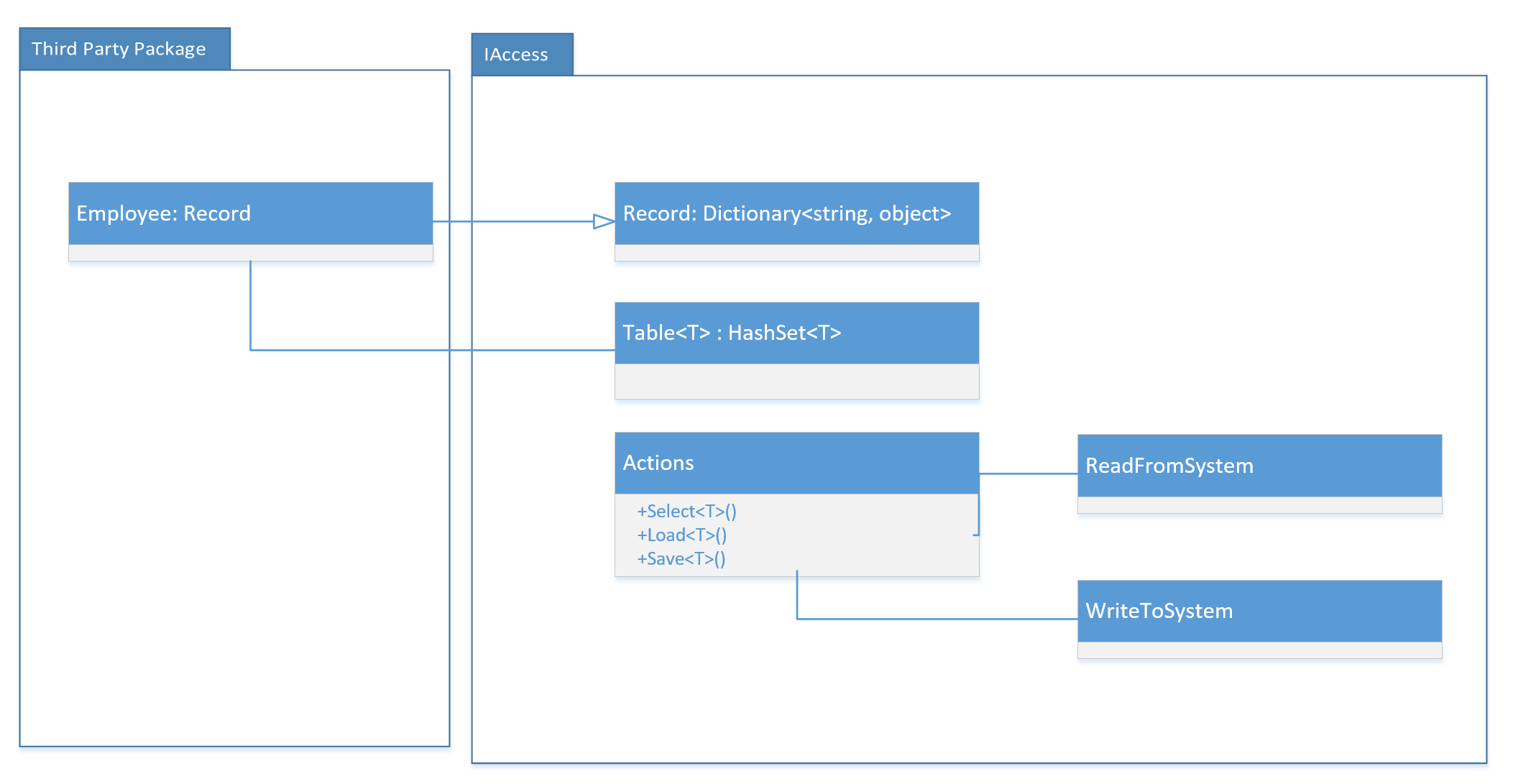


Abbildung 6: UML-Klassendiagram zu iAccess

## Sequenzdiagram

### Erstellen eines neuen Record

Um einen neuen Record erstellen zu können wird zuerst eine Tabelle mit dem Typ des Records erstellt. Anschliessen wir ein neuer Record dieses Typs erstellt und der Tabelle hinzugefügt. Um die Änderung persistent zu speichern, wird die Save Funktion aufgerufen.

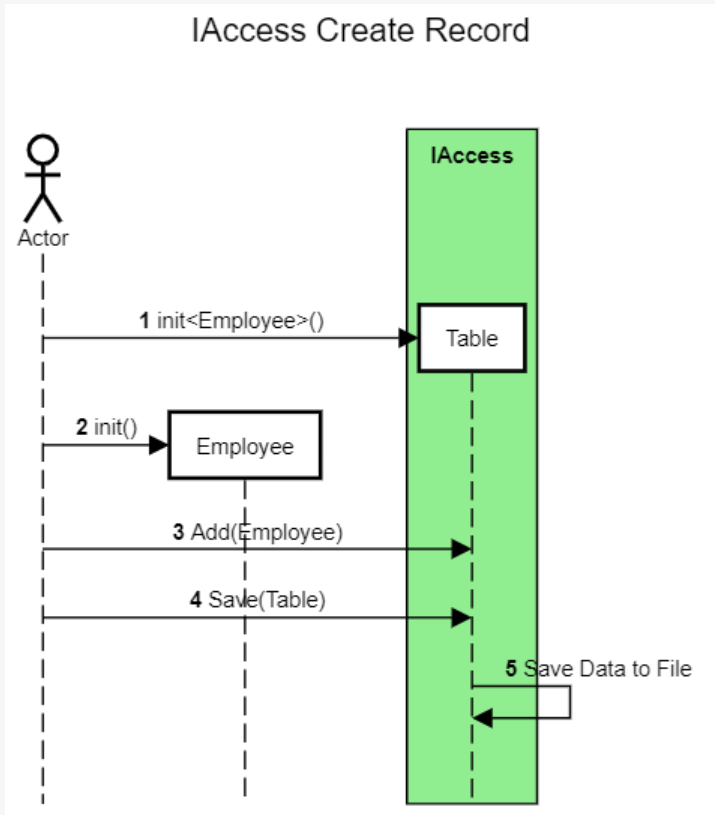


Abbildung 7: UML-Sequenzdiagram zum Erstellen eines neuen Records.

### Select eines Records

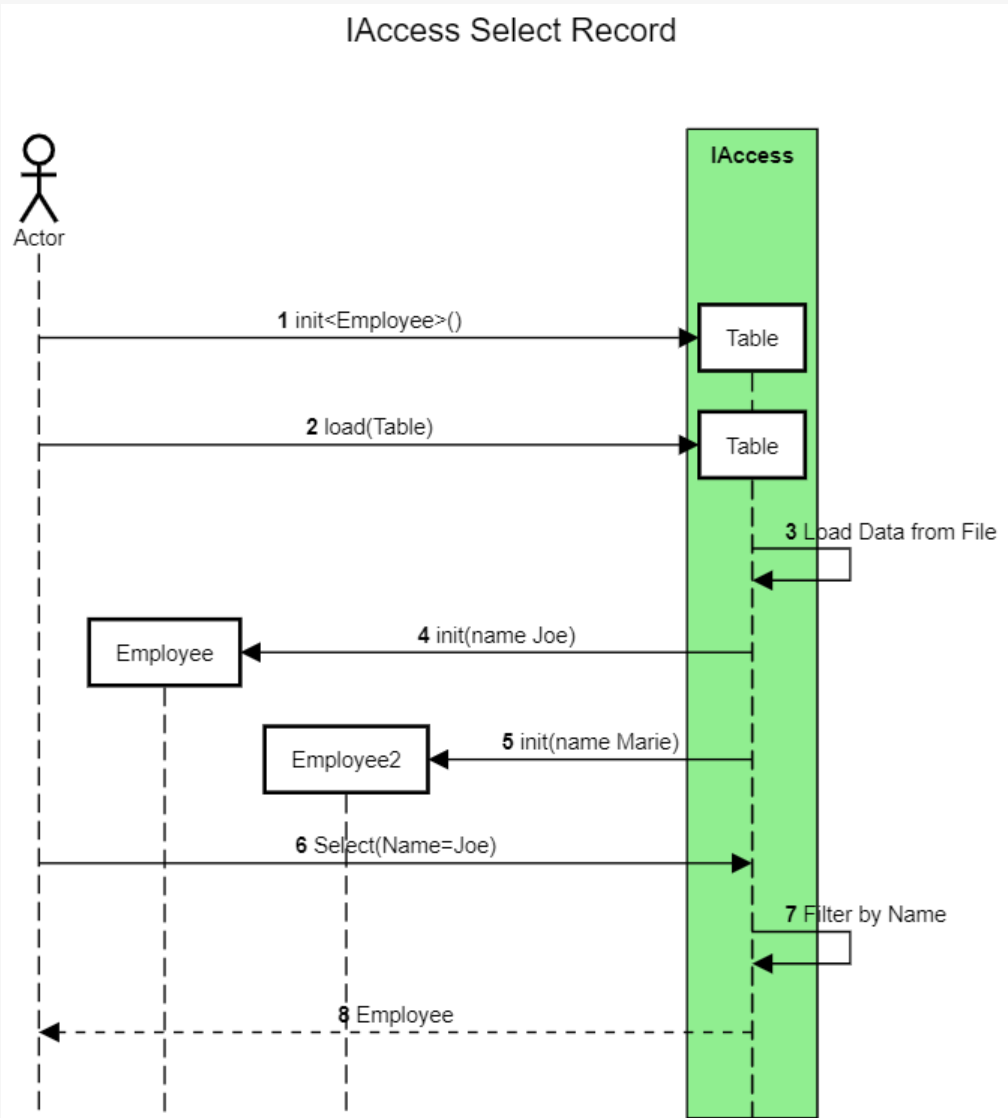
Um einen Record auslesen zu können, müssen zuerst alle Records dieses Typs geladen werden. Dies geschieht über den Generic Typ der Tabelle. Beim auslesen der Daten einer Tabelle können mehrere Predicates mitgegeben werden um die Daten zu filtern.

Abbildung 8: UML-Sequenzdiagram zum Selektieren eines Records.

### Update eines Records

Die Attribute eines ausgelesen Records können direkt angepasst werden. Durch die bestehende Verlinkung mit einer Tabelle werden die Änderung beim persistenten speichern der Tabelle aktualisiert.

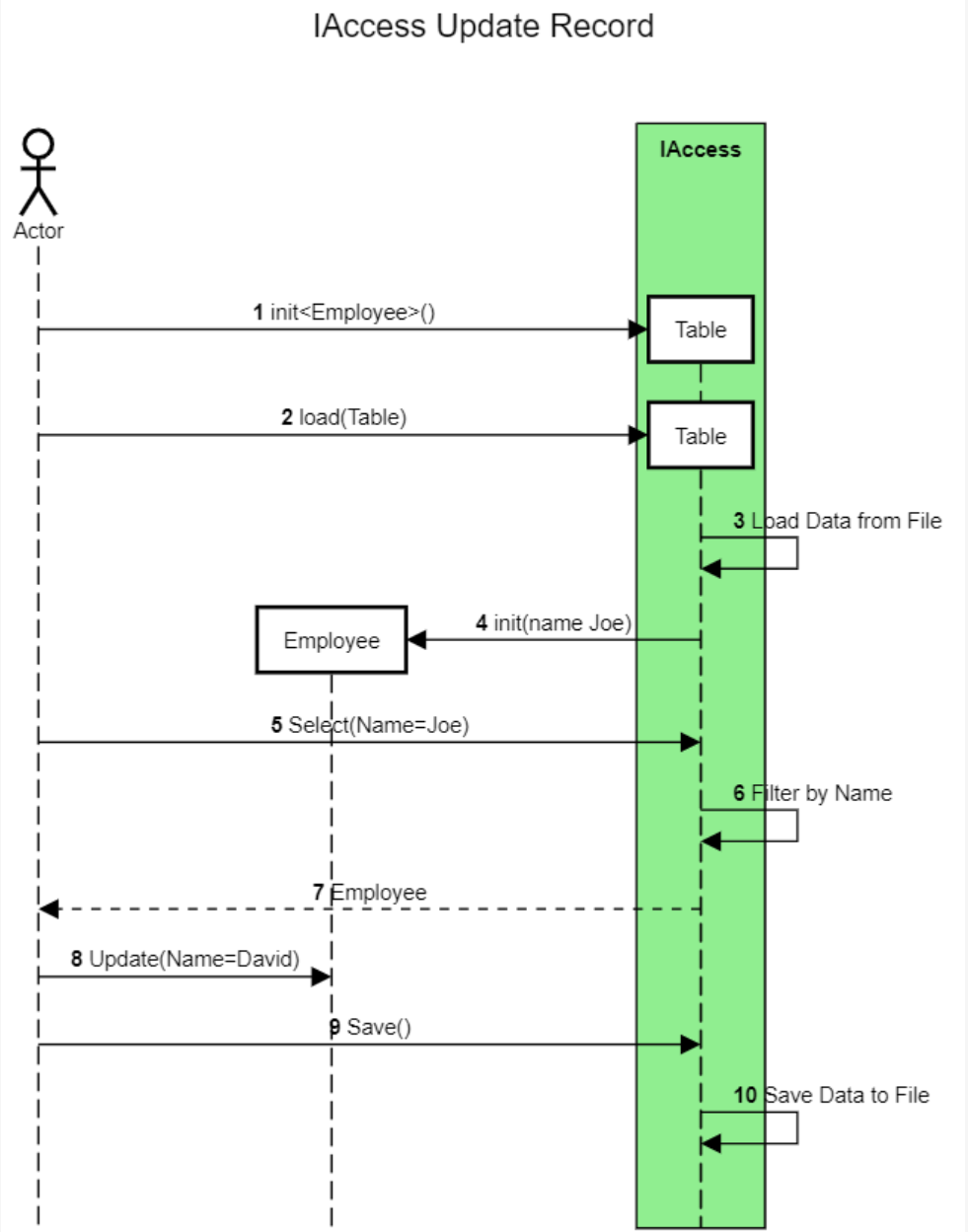


Abbildung 9: UML-Sequenzdiagram zum Updaten eines Records.

### Löschen eines Records

Wir ein Record aus der Tabelle entfernt, wird er beim nächsten Speichern der Tabelle im File überschrieben und ist endgültig gelöscht.

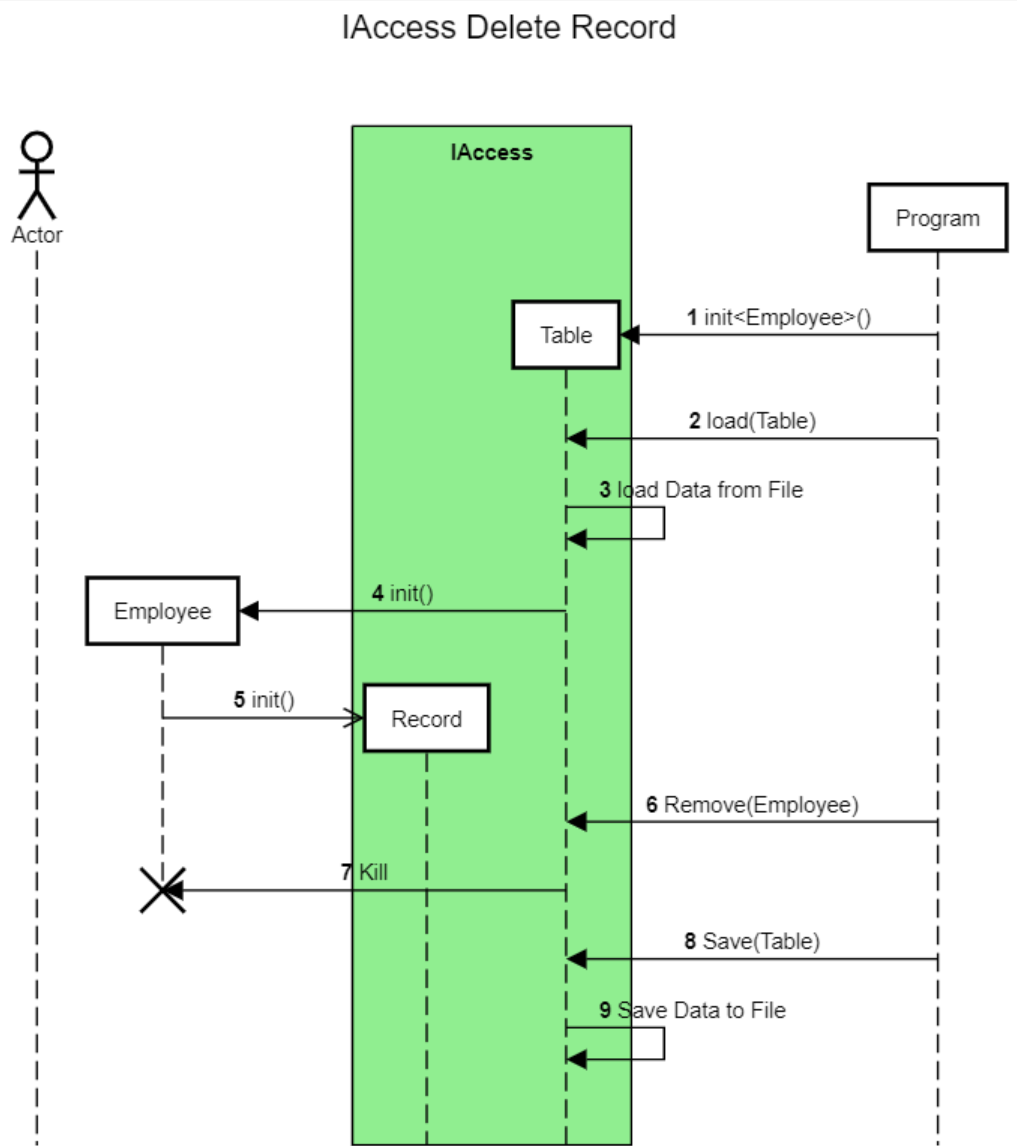


Abbildung 10: UML-Sequenzdiagram zum Löschen eines Records.

# Realisierung

Im nachfolgenden Kapitel wir auf die Methodik der Realisierung eingegangen.

## Versionierung

Sowohl der erstellte Source-Code als auch diese Dokumentation werden mit GIT versioniert. Das Repository wird auf Github als privates Repository gehostet, d.H. nur David Hänni und Severin Gafner haben Zugriff darauf.

## Speichern der Daten

Die erstellen Daten werden in CSV-Dateien abgespeichert. Für jeden Record Typ gibt es eine eigene Datei. Der Dateiname wird automatisch vom Objektnamen übernommen. Die Datei wird erstellt falls sie noch nicht existiert.



Abbildung : Write Code

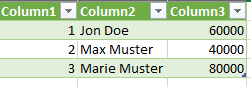


Abbildung : Write CSV

Die Write Methode looped durch alle Einträge in der mitgegebenen Tabelle und fügt Anführungszeichen an falls es sich bei diesem Wert um einen String handelt. Ansonsten wird er normal hinzugefügt. Danach werden Werte pro Record mit einem Komma unterteilt.



Abbildung : Write loop

Zum Schreiben wird die verfügbare C# Funktion «StreamWriter» benutzt. Diesem wird der Pfad angegeben, wo die Datei abgespeichert werden soll. Der Pfad dazu kann in der Datei «Config.cs» angepasst werden. Der zweite Parameter «false» gibt an ob bestehender Inhalt überschrieben werden soll.



Abbildung : StreamWriter

## Lesen der Daten

Die gespeicherten Daten können aus den CSV Dateien ausgelesen werden.



Abbildung : Read Code

Zum Lesen wird die verfügbare C# Funktion «StreamReader» benutzt. Diese benötigt ebenfalls den Path zur Datei, welcher über die Variable im Config File und dem mitgegebenen Objekt Type zusammengesetzt wird.



Abbildung : StreamReader

Die Extrahierten strings werden daraufhin überprüft. Falls es direkt möglich ist sie in einen Integer oder Boolean umzuwandeln soll dies getan werden. Falls nicht, handelt es sich um einen String. Diesem muss nur die Anführungszeichen entfernt werden.



Abbildung : Parsen der gelesen Werte

Danach wird ein Objekt mit dem übergebenen Type erstellt. Dies wird für jede Zeile im CSV gemacht und anschliessend in einen Table geschrieben welcher zum Schluss zurückgegeben wird.



Abbildung :Zurückgeben gelesener Werte

Ein Beispiel für gelesene Werte könnte so aussehen:

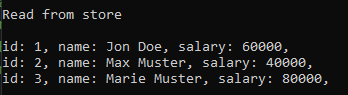


Abbildung : Anzeigen gelesener Werte

# Glossar

|  |  |
| --- | --- |
| **Begriff** | **Beschreibung** |
| C# | Eine Objektorientierte Programmiersprache von Microsoft |
| Record | Eine Zeile in der Tabelle |
|  |  |
|  |  |
|  |  |

Table : Glossar